

EXTENSIBLE MARKUP LANGUAGE RENDERING AND LAYOUT

FIELD OF THE INVENTION

This invention relates generally to rendering and layout functionality in a markup
5 language, such as the HyperText Markup Language (HTML).

BACKGROUND OF THE INVENTION

Markup languages, such as HTML, are currently the primary manner by which
information is exchanged over the worldwide web ("the web") of the Internet, or over
related networks such as intranets and extranets. Markup languages in general provide a
10 relatively rich assortment of functionality. Two types of functionality are especially
relevant to the present invention: rendering behaviors, and layout behaviors. A behavior
generally refers to an action, or processing, that can be performed in relation to a markup
language element. An element in a markup language is a discrete entity that can be the
combination of a set of markup language tags, any content contained between the tags,
15 and any attributes the tags may have. Rendering behaviors generally are behaviors that
specify how markup language elements are to be rendered, or drawn, on a medium, such
as a display screen or on a printed page. Layout behaviors generally are behaviors that
specify how markup language elements are to be laid out, or positioned and sized, on the
medium.

20 Core engines that process markup languages, such as those found in web
browsing computer programs and components of operating systems, do provide a rich
assortment of behaviors for content authors. However, invariably the content authors

may want to provide layout and rendering functionality that are not supported by the core engines. For example, the authors may want to position and size markup language elements in a way that is not supported by the core engine, or may want to draw the elements in a way that is not supported by the core engine. The prior art, however, only provides limited support for extensible behaviors. That is, the prior art only provides limited support for extended layout and rendering behaviors that are not supported by the core engine itself.

One way that extended behaviors are supported in the prior art is by using a technology known as a plug-in. Plug-ins are computer programs that can be called by the core engine to completely take over all the behaviors of an element associated with a plug-in, and that are referenced in HTML code. Plug-ins, while useful, have a significant disadvantage in that they take over processing of the element from the core engine on a wholesale basis. The core engine views a plug-in as a black box meant to process a given element or elements. The core engine does not participate in the processing of elements that are processed by the plug-in. Likewise, the plug-in cannot delegate some of the processing to the core engine. The plug-in represents an all-or-nothing affair: either the plug-in takes complete control of processing an element, without support from the core engine, or it does not. As such, the plug-in does not represent a complete solution to content authors' desire to extend behaviors supported by the core engine.

Another limited solution to provide extended behaviors in the prior art is by using dynamic HTML (DHTML) behaviors, which some prior art web browsing computer programs and operating system components support. DHTML behaviors are written similar to scripts, except that the DHTML behaviors can be separated from the HTML

viewed as a plug-in-like technology, but which is based on software objects, instead of software programs, as plug-ins are.

Content authors, therefore, are currently presented with prior art extensibility of behaviors that may not represent how they actually want to extend those behaviors.

SUMMARY OF THE INVENTION

To extend a comparable behavior of the core engine with a behavior of an external component, the mechanism does the following. It calls a behavior initialization method of the external component to learn how the external component's behavior wants to participate with the core engine's comparable behavior. Once this is known, the

5 mechanism then calls a behavior method of the external component that provides the behavior when the core engine is providing the comparable behavior. As a result, the external components' behavior participates with the core engine's comparable behavior. In addition, the mechanism may receive calls from the external component to a corresponding behavior method of the mechanism. This allows the external component

10 to communicate with the core engine during participation of the external component's behavior with the core engine's comparable behavior.

The invention provides for advantages not found within the prior art. Complete participation with existing behaviors can be accomplished by the extended behaviors of external components of the invention. This is in distinction with DHTML behaviors,

15 which can only accomplish such participation on a limited basis, to supplement in a minor way or group existing behaviors. Furthermore, this is completely unlike extended behaviors that are supported by plug-ins or OLE objects, which cannot participate with existing behaviors at all. As a result, the extended behaviors provided by the invention give content authors a more complete solution to existing behavior extensibility. New

20 behaviors can participate with existing behaviors as much or as little as is desired. The authors are not locked into providing only limited participation, as they are with DHTML behaviors, or never participating with existing behaviors, as they are with plug-ins and OLE objects.

shown by way of illustration specific exemplary embodiments in which the invention may be practiced. These embodiments are described in sufficient detail to enable those skilled in the art to practice the invention. Other embodiments may be utilized, and logical, mechanical, electrical, and other changes may be made without departing from the spirit or scope of the present invention. The following detailed description is, therefore, not to be taken in a limiting sense, and the scope of the present invention is defined only by the appended claims.

How Extended Behaviors Are Supported by the Invention

FIG. 1 is a diagram of a system 100 showing how the prior art processes markup language code, such as HTML markup language code. The markup language code 104 references behaviors, such as layout and rendering behaviors, supported by the core engine 102. The markup language code 104 can be, for example, an HTML that represents a web page of a web site on the Internet, or on an intranet or extranet. The core engine 102 processes the elements and otherwise processes the markup language code 104, resulting in output on an output medium 106. The core engine 102 can be part of a web browsing computer program or operating system component. The output medium 106 can be, for example, a display monitor, a printed page output by a printer, or another type of output medium.

FIG. 2 is a diagram of a system 200 showing how markup language processing is accomplished by an embodiment of the invention, including extended behaviors supported by external components. The markup language code 104 is again processed by the core engine 102. However, for behaviors referenced in the markup language code 104 that are not supported by the core engine 102, a mechanism 202 exists that passes

laid out, or positioned and sized, on the medium 106. Each of these behaviors can be attached behaviors or element behaviors. Attached behaviors are attached in a non-permanent manner to a markup language element. An element in a markup language is a discrete entity that can be any content contained between the tags, and any attributes the tags may have. Such behaviors can be applied, or mapped, to an element, and subsequently removed from the element. Conversely, element behaviors are permanently attached to a markup language element. They cannot be later removed once applied to an element.

An example of how a behavior supported by the external component 204 can be an attached behavior applied and then unapplied to an HTML element is:

```
<DIV STYLE = "behavior: url(http://www.abc.com/object.dll);" >  
</DIV>
```

In this example, the behavior is implemented by the external component "object.dll". This behavior is attached to the HTML element, and later unattached to the HTML element.

An example of how a behavior supported by the external component 204 can be an element behavior permanently applied to an HTML element is:

```
<?import namespace="foo" implementation="http://www.abc.com/object.dll">  
<foo:bar>  
</foo:bar>
```

In this example, "foo" is identified as a namespace, such that all the tags associated with this namespace are implemented by the external component "object.dll". This behavior is attached to the element "bar" in the "foo" namespace. As an element behavior, it cannot be unattached to the HTML element once it has been attached.

09677403-093000

FIG. 3 is a diagram of a system 300 showing in more detail how markup language code can reference an external component 204 that supports an extended behavior. The markup language code 104 specifically has a body component 302, a div component 304, and a paragraph component 306. The body component 302, the div component 304, and the paragraph component 306 have no externally defined behavior and are handled by the core engine 102. The paragraph component 306 references a peer holder 308 within the mechanism 202. The peer holder 308 accesses the interface 206 of the external component 204, and likewise exposes the interface 208 that is accessed by the external component 204.

FIG. 4 is a flowchart of a method 400 showing how the mechanism 202 provides for extended behaviors supported by external components, such as the external component 204, which is specifically used for the description of the method 400 as an example. In 402, the mechanism 202 calls a query interface of the external component 204. The query interface is a standard interface, which the mechanism 202 calls to obtain a pointer to the first interface 206 exposed by the external component 204. In addition, the mechanism 202 passes a pointer to the second interface 208 that it exposes, to the external component 204, by the query interface call.

The mechanism then calls a behavior initialization method of the first interface 206 in 404. In response to this call, the external component 204 indicates the type of extended behavior that it is supporting, either a layout behavior or a rendering behavior. The external component 204 also indicates how this behavior of the component 204 will participate with the comparable behavior of the core engine 102. The comparable behavior of the core engine 102 is the behavior of the core engine 102 that the extended

behavior supported by the external component 204 is replacing, supplementing, complementing, or to which the extended behavior otherwise relates.

Where the behavior supported by the external component 204 is a layout behavior, participation of this behavior with the comparable behavior of the core engine 102 is preferably in one of two ways. First, the behavior can be fully delegated to the external component 204 from the core engine 102. Second, the behavior implemented by the external component 204 can be called after the comparable behavior of the core engine 102 is performed. A complete example of such layout behaviors is presented in a subsequent section of the detailed description.

Where the behavior supported by the external component 204 is a rendering behavior, participation of this behavior with the comparable behavior of the core engine 102 is also preferably in one of two ways. First, the rendering by the behavior of the external component 204 can completely replace rendering by the comparable behavior of the core engine 102. Second, the rendering by the behavior of the external component 204 can be interspersed with rendering by the comparable behavior of the core engine 102. A complete example of such rendering behaviors is also presented in a subsequent section of the detailed description.

As necessary, the mechanism calls a behavior method of the first interface 206 in 406. The behavior method is called when the core engine 102 is providing the comparable behaviors to the behavior of the external component 204. This is done so that the behavior of the external component 204 can participate in processing with the comparable behaviors of the core engine 102. Similarly, the mechanism receives calls from the external component 204 to a corresponding behavior method of the second

interface 208 in 408. When the external component 204 needs to submit information to the core engine 102, it calls this method of the second interface 208. This is also done so that the behavior of the external component 204 can properly participate with the comparable behaviors of the core engine 102. Examples of behavior methods of the first interface 206 and of corresponding behaviors methods of the second interface 208 are described in subsequent sections of the detailed description, with respect to layout and rendering behaviors.

Rendering Behaviors

Extended rendering behaviors supported by the external component 204 are described in this section of the detailed description. FIG. 5 is a diagram showing how the core engine 102 divides rendering among a number of layers, and how the behavior of the external component 204 can participate with the behaviors of the core engine 102. There are layers 502, 504, 506, and 508. The layers 502, 504, and 506 are referred to as content layers. The flow layer 504 is the layer on which text is drawn. There can be a number of layers above and below the flow layer 504, such as the positive z layer 502 above the flow layer 504, and the negative z layer 506 below the flow layer 504. Below all of these layers is the background layer 508.

Rendering behaviors supported by the external component 204 can interact with the different layers of the diagram 500 in two primary ways. First, the behaviors can intersperse their drawing with the rendering performed by the behaviors of the core engine 102. Second, the behaviors can effect wholesale replacement of the drawing process from the behaviors of the core engine 102. With respect to the former, the rendering behaviors supported by the external component 204 can draw below the

content, after the background layer 508 is drawn, as indicated by the arrow 520. The rendering behaviors can also draw below the flow layer 504, where the behavior is called after the elements with a negative z-index are drawn, as indicated by the arrow 518. The rendering behaviors can draw above the flow layer, after the elements of the low are drawn, as indicated by the arrow 516. Finally, the rendering behaviors can draw above the content, after the elements with a positive z-index are drawn, as indicated by the arrow 514.

With respect to wholesale replacement of the drawing process, the rendering behaviors supported by the external component 204 can replace the drawing of all the layers 502, 504, 506, and 508, as indicated by the curved bracket 522. The rendering behaviors can also replace just the content layers 502, 504, and 506, as indicated by the curved bracket 510. Finally, the rendering behaviors supported by the external component 204 may replace only the background layer 508, as indicated by the arrow 512. Two options not indicated in the diagram 500 of FIG. 5 include, first, rendering behaviors may have a “none” setting, in which they do not affect the drawing process of the behaviors of the core component 102 at all. Second, there can be a “window top” setting by which the behavior supported by the external component 204 is called after the entire markup language page has been drawn, such that the behavior draws on top of all the content on the page.

As has been described in conjunction with FIGs. 2 and 3, there is a pair of interfaces 206 and 208 for communication between the external component 204 and the mechanism 202 that can be a part of the core engine 102. With respect to rendering behaviors, the interface 206 exposed by the external component 204 is referred to as

IHTMLPainter, whereas the interface 208 exposed by the mechanism 202 is referred to as IHTMLPaintSite. The external component may also expose two other interfaces 206, referred to as IHTMLPainterEventInfo, and IHTMLPainterOverlay. Each of these interfaces is described in turn.

5 The IHTMLPainter interface is specified as follows.

```

IHTMLPainter
Draw(
[in] RECT rcBounds      Bounds of the object relative to the DC's origin
[in] RECT rcUpdate      Bounds of the update region relative to the DC's origin
10 [in] LONG lDrawFlags   Flags affecting drawing
[in] HDC hdc            The device context handle (or NULL)
[in] LPVOID pvDrawObject Pointer to a surface object (or NULL)
);

15 GetPainterInfo(
[out] HTML_PAINTER_INFO* pInfo Characteristics of the rendering behavior
);

This structure allows the behavior to convey information to the core engine. The fields are defined as
20 follows.

typedef struct _HTML_PAINTER_INFO
{
25     LONG lFlags;
    LONG lZOrder;
    IID iidDrawObject;
    RECT rcExpand;
} HTML_PAINTER_INFO;

30 HitTestPoint(
[in] POINT pt,          Point in coordinates relative to the rcBounds origin where the hit occurs
[out] BOOL *pbHit       Returns true if the element is actually hit
[out] LONG *plPartID    returns a part ID number, indicating which part of the element was hit. If an
event is fired as a result of this hit-test, the part ID is made available in the event object
35 );

OnResize( [in] SIZE size);      called whenever the element is resized, and informs the component of
the new size.

```

40 The IHTMLPainter interface has four methods: Draw, GetPainterInfo, HitTestPoint, and OnResize. The Draw method corresponds to what has been referred to as the behavior method of the first interface 206 in 406 of FIG. 4. The GetPainterInfo method corresponds to what has been referred to as the behavior initialization method of the first

interface 206 in 404 of FIG. 4. The HitTestPoint method and the OnResize method also correspond to what has been referred to as the behavior method of the first interface 206 in 406 of FIG. 4. Each of these methods is now described in turn.

With respect to the GetPainterInfo method, the mechanism 202 that can be a part of the core engine 102 calls this method to determine how the behavior of the external component 204 desires to participate in the rendering process. In particular, the external component 204 returns information in the structure HTML_PAINTER_INFO. The IFlags are a combination of the following.

| | | |
|----|----------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 10 | HTMLPAINTER_OPAQUE = 0x000001, rectangle | the component paints every pixel in its drawing |
| | HTMLPAINTER_TRANSPARENT = 0x000002, lower content to show through | the component may not draw every pixel, allowing |
| | HTMLPAINTER_ALPHA = 0x000004, content | the component does alpha-blending with lower |
| 15 | HTMLPAINTER_COMPLEX = 0x000008, content | the component does full compositing with lower |
| | HTMLPAINTER_OVERLAY = 0x000010, draw its content | the component uses a hardware overlay buffer to |
| 20 | HTMLPAINTER_HITTEST = 0x000020, HTMLPAINTER_SURFACE = 0x000100, HTMLPAINTER_3DSURFACE = 0x000200, HTMLPAINTER_NOBAND = 0x000400, bands | the component wishes to participate in hit-testing the component draws to a surface the component draws to a 3D surface the component is not prepared to draw its content in |
| 25 | HTMLPAINTER_NODC = 0x001000, HTMLPAINTER_NOPHYSICALCLIP = 0x002000, the DC; it does its own clipping | the component does not need a DC the component does not need a physical clip set on |
| | HTMLPAINTER_NOSAVEDC = 0x004000, during drawing; calls to Draw() do not need to be bracketed by SaveDC() and RestoreDC() | the component does not change the state of the DC |
| 30 | HTMLPAINTER_SUPPORTS_XFORM = 0x008000, transformation while drawing | the component is prepared to use a complex |
| | HTMLPAINTER_EXPAND = 0x010000 rcExpand | the component wishes the core engine to honor the |
| 35 | HTMLPAINTER_NOSCROLLBITS = 0x020000 bits, but instead must ask the component to redraw | the core engine cannot scroll this element by copying |

The surfaces referred to can be DirectDraw surfaces, as known within the art. The DC referred to is a device context, as known within the art. The LZOrder indicates the drawing layer at which the Draw method should be called. This can be one of the following.

calls this method when its picture is out of date, as specified by a rectangle. This invalidates its rectangle, and forces the mechanism 202 that can be part of the core engine 102 to force a draw. With respect to the InvalidateRegion method, the external component 204 similarly calls this method when its picture is out of date, but as specified
 5 by a non-rectangular region.

With respect to the GetDrawInfo method, the external component 204 calls this method to receive the additional information that has been indicated as available when the mechanism 202 called the Draw method of the external component 204. The LFlags can be set a combination of the following:

- 10 HTMLPAINT_DRAWINFO_VIEWPORT = 0x000001, return the rcViewport
- HTMLPAINT_DRAWINFO_UPDATEREGION = 0x000002, return the hrgnUpdate
- HTMLPAINT_DRAWINFO_XFORM = 0x000004, return the transform

The rcViewPort is the part of a region that is currently in view. For example, when the
 15 scrollable, the part in the window that is currently visible is specified by the rcViewPort. This is useful to the external component 204 if it wishes to draw a fixed background, such as a watermark, behind scrollable content. hrgnUpdate is the update region, describing the minimal surface area that needs repainting, and is contained within the rcUpdate passed to the Draw method. xform is the transformation matrix that must be
 20 applied to all drawing coordinates before rendering. This is an affine linear transformation in the following format.

```
typedef struct HTML_PAINT_XFORM
{
  25   float eM11;
  float eM12;
  float eM21;
  float eM22;
  float eDx;
  float eDy;
  30 }
```


Layout Behaviors

Extended layout behaviors supported by the external component 204 are described in this section of the detailed description. The behaviors can use the interfaces described in this section to specify their size and position. Furthermore, the behaviors can use the interfaces described in this section to modify their natural size in order to have more or less space allocated to them.

Layout behaviors supported by the external component 204 can interact with the natural sizing of elements that is normally performed by the core engine 102 in three ways. First, layout can be completely delegated to the external component 204. Second, layout can be modified by the external component 204 after the core engine has proposed a natural size for the element. Third, the layout behavior can specify that it is also a rendering behavior that needs additional space outside the element's borders.

As has been described in conjunction with FIGs. 2 and 3, there is a pair of interfaces 206 and 208 for communication between the external component 204 and the mechanism 202 that can be a part of the core engine 102. With respect to layout behaviors, the interface 206 exposed by the external component 204 is referred to as IElementBehaviorLayout, whereas the interface 208 exposed by the mechanism 202 is referred to as IElementBehaviorSiteLayout. Each of these interfaces is described in turn.

The IElementBehaviorLayout interface is specified as follows.

```

20 IElementBehaviorLayout {
    GetLayoutInfo( LONG * pIFlags );

    GetSize( [in] long lCalcMode,
25         [in] SIZE sizeContent,
         [in, out] POINT *ptOffset,
         [in, out] POINT *ptOffset,
         [in, out] SIZE *psize );

    GetPosition( [in] LONG lFlags,

```

```

[in, out] POINT* pptTopLeft);

MapSize( [in] SIZE * psizeIn,
         [out] RECT * prcOut);
5  }

```

The interface has four methods: GetLayoutInfo, GetSize, GetPosition, and MapSize. The GetLayoutInfo method corresponds to what has been referred as the behavior initialization method of the first interface 206 in 404 of FIG. 4. The other methods

10 correspond to what has been referred to as the behavior method of the first interface 206 in 406 of FIG. 4. Each of these methods is described in turn.

With respect to the GetLayoutInfo method, the mechanism 202 that can be a part of the core engine 102 calls this method to determine whether the external component 204 wants to participate in layout participation either before and/or after natural sizing

15 occurs by the core engine 102. There are three values that are recognized, as follows.

0x01 - FULLDELEGATION -- the behavior completely owns sizing
 0x02 - MODIFYNATURAL -- the behavior wants to modify the natural-content-size/ proposed-size
 0x04 - MAPSIZE -- the behavior is also a rendering behavior that needs additional space outside the borders. MapSize() is only called if this flag is turned on.

20

If FULLDELEGATION is specified, then MODIFYNATURAL is ignored.

With respect to the GetSize method, the mechanism 202 calls this method when it needs to determine the size of the element controlled by the component 204. The LCalcMode parameter passed to this method is used to determine the layout mode, which

25 can be minimum, min-max, or natural sizing passes. These enumerated values describe the various layout passes the layout may put its element through. There are the following layout calculation modes.

0x0001 - BEHAVIORLAYOUTMODE_NATURAL - The core engine is asking for the natural size of the element. Both width and height must be returned.

depending on whether the output medium is, for example, a display screen, as opposed to a page printed by a printer.

Example Computerized Device

The invention can be implemented within a computerized environment having one or more computerized devices. The diagram of FIG. 6 shows an example computerized device 600. The computerized device 600 may be the device in which the system 200 of FIG. 2 is implemented. The example computerized device 600 can be, for example, a desktop computer, a laptop computer, or a personal digital assistant (PDA).

The invention may be practiced with other computer system configurations as well, including multiprocessor systems, microprocessor-based or programmable consumer electronics, network computers, minicomputers, and mainframe computers. The invention may be practiced in distributed computing environments where tasks are performed by remote processing devices that are linked through a communications network.

The device 600 includes one or more of the following components: processor(s) 602, memory 604, storage 606, a communications component 608, input device(s) 610, a display 612, and output device(s) 614. For a particular instantiation of the device 600, one or more of these components may not be present. For example, a PDA may not have any output device(s) 614. The description of the device 600 is to be used as an overview of the types of components that typically reside within such a device, and is not meant as a limiting or exhaustive description.

The processor(s) 602 may include a single central-processing unit (CPU), or a plurality of processing units, commonly referred to as a parallel processing environment.

The memory 604 may include read-only memory (ROM) and/or random-access memory (RAM). The storage 606 may be any type of storage, such as fixed-media storage devices and removable-media storage devices. Examples of the former include hard disk drives, and flash or other non-volatile memory. Examples of the latter include tape drives, optical drives like CD-ROM drives, and floppy disk drives. The storage devices and their associated computer-readable media provide non-volatile storage of computer-readable instructions, data structures, program modules, and other data. Any type of computer-readable media that can store data and that is accessible by a computer can be used.

The device 600 may operate in a network environment. Examples of networks include the Internet, intranets, extranets, local-area networks (LAN's), and wide-area networks (WAN's). The device 600 may include a communications component 608, which can be present in or attached to the device 600. The component 608 may be one or more of a network card, an Ethernet card, an analog modem, a cable modem, a digital subscriber loop (DSL) modem, and an Integrated Services Digital Network (ISDN) adapter. The input device(s) 610 are the mechanisms by which a user provides input to the device 600. Such device(s) 610 can include keyboards, pointing devices, microphones, joysticks, game pads, and scanners. The display 612 is how the device 600 typically shows output to the user. The display 612 can include cathode-ray tube (CRT) display devices and flat-panel display (FPD) display devices. The device 600 may provide output to the user via other output device(s) 614. The output device(s) 614 can include speakers, printers, and other types of devices.

The methods that have been described can be computer-implemented on the device 600. A computer-implemented method is desirably realized at least in part as one or more programs running on a computer. The programs can be executed from a computer-readable medium such as a memory by a processor of a computer. The
5 programs are desirably storable on a computer or machine-readable medium, such as a floppy disk or a CD-ROM, for distribution and installation and execution on another computer. The program or programs can be a part of a computer system, a computer, or a computerized device.

Conclusion

10 It is noted that, although specific embodiments have been illustrated and described herein, it will be appreciated by those of ordinary skill in the art that any arrangement that is calculated to achieve the same purpose may be substituted for the specific embodiments shown. This application is intended to cover any adaptations or variations of the present invention. Therefore, it is manifestly intended that this invention
15 be limited only by the claims and equivalents thereof.

00967403-093000